UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de Sistema de Navegación de Robots basado en Adquisición Óptica

Jorge Real García
Tutor: Kostadin Koroutchev

MAYO 2019

Desarrollo de Sistema de Navegación de Robots basado en Adquisición Óptica

AUTOR: Jorge Real García

TUTOR: Kostadin Koroutchev

Dpto. Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo de 2019

Resumen

Este Trabajo Fin de Grado propone un nuevo sistema de localización en tiempo real sin precisar del uso de gps o balizas.

La idea de este proyecto se basa en la patente 2566427, en la que se demuestra cómo es posible posicionar un dispositivo en una superficie heterogénea con la previa elaboración de un mapa mediante un barrido del área implicada.

Para ello se ha construido un robot móvil a partir de una Raspberry Pi, propulsado por dos servomotores y manejado mediante control remoto, capaz de identificar las coordenadas en las que se encuentra sobre un área escaneada con la ayuda de una cámara.

Se trata de un sistema innovador, puesto que se ayuda exclusivamente de un sensor óptico, evitando el uso de mediciones láseres o detectores de objetos como sí utilizan otros dispositivos del mercado.

El prototipo construido es una muestra a pequeña escala del alcance de ésta premisa, dado que a pesar del reducido tamaño tanto del móvil como del mapa elaborado para las pruebas, el algoritmo codificado podría incorporarse a vehículos más robustos o incluso drones.

El objetivo de este sistema es lograr obtener información sobre la situación de un robot en áreas con acceso restringido o circunstancias desfavorables, en las cuales los sistemas de posicionamiento existentes en la actualidad no funcionarían. Algunas de las situaciones en las que podría verse el potencial de este vehículo sería en la localización dentro de edificaciones o en zonas en las que existan inhibidores de señal.

Palabras clave

Localización, posicionamiento, robot, prototipo, sensor óptico.

Abstract

This Final Degree Project proposes a new real-time location system without specifying the use of GPS or beacons.

The idea of this project is based on the patent 2566427, which shows how to position a device on a heterogeneous surface based only on optical information and previous elaboration of a map by scanning the involved surface.

To implement this method, a mobile robot has been built. The robot is controlled by a Raspberry Pi and is powered by two servomotors and managed by remote control, and identifies its position on a previously scanned area with by using only the information of an optical on-board camera.

It is an innovative system, because the localization it is exclusively based on an optical sensor (standard camera), avoiding the use of laser measurements, object detectors or external beamers.

The built prototype is a small-scale sample of this premise´s scope, given that despite the small size of both the mobile robot and the elaborated map prepared for testing, the algorithm could be incorporated on more sophisticated vehicles or drones.

The goal of this system is to obtain information about the position of the robot in restricted access areas or unfavorable circumstances, where current positioning systems would not work. The potential of this method could be seen inside buildings or in areas with signal inhibitors.

Keywords

Location, positioning, robot, prototype, optical sensor.

Agradecimientos

En primer lugar quería agradecer a mi tutor, Kostadin Koroutchev, todo el tiempo que me ha dedicado para sacar este proyecto adelante. Ademas, le animo a que siga proponiendo trabajos tan originales como este, siempre encontrará alumnos con ganas de hacer algo diferente.

También quiero dar las gracias a mi familia, mis amigos y mi pareja, los cuales me han dado las fuerzas y el apoyo necesario para no rendirme con este reto.

Finalmente quiero agradecer a la Universidad Autónoma de Madrid y a sus profesores todos estos años de enseñanzas, pues han servido para formarme en la disciplina que siempre quise y a la que, por suerte, me dedico actualmente.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1.Introducción

1.1. Motivación

Hoy en día existen multitud de sistemas de localización en tiempo real, como los basados en satélites, radiofrecuencia, infrarrojos o balizas. Sin embargo, cuando se trata de posicionamiento en interiores los anteriores sistemas no funcionan de forma tan eficaz o precisan de sistemas externos.

Es por esto, por lo que surge la idea de buscar un sistema alternativo, autónomo, y que no precise de emisores artificiales externos. Éste debe ser capaz de obtener su posición basándose en las imágenes obtenidas con una cámara, capturando únicamente la superficie recorrida, sin tener en cuenta los objetos u obstáculos del entorno.

1.2. Objetivos

El objetivo de este Trabajo de Fin de Grado es llevar a la práctica lo detallado en la patente 2566427, propiedad de la Universidad Autónoma de Madrid. En la que se demuestra la posibilidad de realizar un posicionamiento óptico de un dispositivo basándose en las características de la textura de una superficie.

Para ello se debe construir un robot manejado por control remoto capaz de enviar las coordenadas en las que se encuentra sobre un superficie aleatoria mapeada previamente. Tan solo podrá ayudarse de una cámara como único sensor y dos motores para el desplazamiento. Además, contará con un adaptador wifi para el enviar los comandos e imágenes y recibir los resultados.

1.3. Organización de la memoria

El presente documento trata de definir todo el ciclo de trabajo de este proyecto concreto y por eso la estructura predefinida ha sufrido algunas variaciones:

- **1.Introducción:** Motivos. Objetivos. Organización de la memoria. Tecnologías empleadas.
- **2. Estado del Arte.**
- **3. Descripción del sistema:** Hardware. Software.
- **4. Documentación técnica:** Aplicación WEB. Motores. Cámara. Preproceso de imágenes. Proceso y algoritmo de generación del mapa. Algoritmo de búsqueda.
- **5. Integración, pruebas y resultados.**
- **6. Conclusiones y trabajo futuro.**

1.4. Tecnologías empleadas

En esta sección se detallan los distintos lenguajes y tecnologías utilizados en el trabajo.

Tecnología	Descripción
Python	Lenguaje de programación multiparadigma.
PyCharm	Entorno de desarrollo integrado para el lenguaje de programación Python, con análisis de código y depurador gráfico.
OpenCV 3.4.3	Biblioteca libre de visión artificial utilizada en el tratamiento de imágenes.
Apache	Servidor Web HTTP.
HTML	Lenguaje de programación para la elaboración de páginas web.
PHP	Lenguaje de programación para el desarrollo web incrustado con HTML.
Sublime Text	Editor de texto y código fuente.
Picamera	Librería de Python para el manejo del módulo de cámara en la Raspberry Pi
Pololu DRV8835	Driver para el manejo del puente H al que se enganchan los motores.

2.Estado del Arte

Los sistemas de localización forman parte de nuestro día a día. Para algunas personas resultan indispensables en su vida. Cuentan con un gran número de aplicaciones, desde la navegación de automóviles o peatones, hasta el rastreo de vehículos o guiado de misiles.

Los existentes en la actualidad pueden clasificarse en dos grandes grupos, los basados en emisores y los que se basan en emisiones propias.

Algunas de las tecnologías basadas en emisores serían la localización mediante satélites, por radiofrecuencia, por infrarrojos o la utilizada en aparatos aeronáuticos (VOR).

La localización por satélite cuenta con tres grandes sistemas, como son:

- GPS: sistema americano, es el más extendido en el mercado debido a su precisión, rapidez y bajo coste.
- Galileo: desarrollado por la Unión Europea para uso civil. Su coste es mucho más elevado que el del GPS, motivo por el cual permanece a la sombra del sistema americano.
- GLONASS: elaborado por la Unión Soviética a finales del siglo XX con carácter militar. Actualmente solo es utilizado en Rusia.
- Beidou: desarrollado por China. Permanece operativo desde el 2000 y ofrece servicios públicos y privados, siendo estos segundos mucho más precisos. Cabe destacar que a diferencia de los anteriores sistemas este tan solo opera en una zona delimitada, con la cobertura suficiente para cubrir la República Popular de China.

Los sistemas de localización basados en radiofrecuencia precisan de la instalación de una serie de etiquetas con antena que servirán de identificador para la máquina que las lea.

Un ejemplo sería el *Cricket*, desarrollado por ingenieros del MIT.

El posicionamiento mediante señales infrarrojas cuenta con muy poco alcance y es usado tan solo en interiores. Se basa en unos emisores de señal ejerciendo de balizas y un dispositivo receptor que triangula su posición a partir de estas.

Los vehículos aeronáuticos utilizan sus propios sistemas de localización, VOR, NDR, ILS, entre otros, para seguir en vuelo la ruta preestablecida y facilitar el aterrizaje. Existen estaciones en posiciones determinadas en cada aeropuerto y en diversos puntos de las rutas encargadas de emitir una radio señal modulada, la cual será recibida por la aeronave. Utilizando código Morse el piloto será capaz de identificar la estación emisora.

En contraposición, existen otros sistemas que basan su posicionamiento en señales emitidas por ellos mismos.

A este grupo pertenecerían los dispositivos que detectan su posición en función de los obstáculos hallados mediante señales ópticas, acústicas o basadas en radiofrecuencias.

Un claro ejemplo de dispositivo basado en estas tecnologías serían los famosos robots aspiradores. Estos sistemas utilizan sus sensores para evitar chocarse, al mismo tiempo registran un mapa virtual, el cual les permitirá desempeñar su labor en futuras ocasiones con más eficacia.

Todos los anteriores sistemas citados comparten una característica, dependen de ayuda externa a ellos, ya sea satélites, balizas u obstáculos que marquen el camino.

El trabajo expuesto pretende dar una nueva solución a este problema, cada vez más demandada. Debe cumplir con todos los requisitos que se le solicitan: precisión, velocidad y, sobre todo, autonomía, cualidad que le haría único en el sector.

3.Descripción del sistema

En esta sección se describe el diseño hardware y software del proyecto, incluyendo la arquitectura de sistema escogida.

3.1. Hardware

La construcción del hardware se ha basado en una Raspberry Pi 2 con el sistema operativo Raspbian basado en Linux. Ésta se aloja sobre una estructura metálica con perforaciones, la cual fue aprovechada de anteriores proyectos de robótica.

Para posibilitar el desplazamiento se han instalado cuatro ruedas, las dos traseras movidas por dos servomotores, controlados con un módulo complementario, el Pololu DRV8835 Dual Motor Driver Kit, el cual ejerce de punte H. Este accesorio se conecta a los gpio de la Raspberry y permite invertir el voltaje en los polos de los dos servos. Con la ayuda de los drivers proporcionados por el fabricante se ha escrito un programa con el que manejar la velocidad y dirección del coche.

Sobre la plataforma también se ha acoplado un portapilas con capacidad para 4 pilas AA que alimenta los dos motores.

El suministro de energía a la CPU se realiza a través de un power Bank USB de 5000mAh, proporcionándole una autonomía de aproximadamente 6 horas.

Para finalizar, el robot cuenta con el módulo pi camera, el cual ejercerá de sensor dado que será el encargado de capturar las imágenes que se analizarán en el proceso de posicionamiento en tiempo real.

Para fijarla se ha construido una estructura con piezas del juguete Meccano a modo de soporte, manteniéndola a una distancia aproximada de 14 cm del suelo.

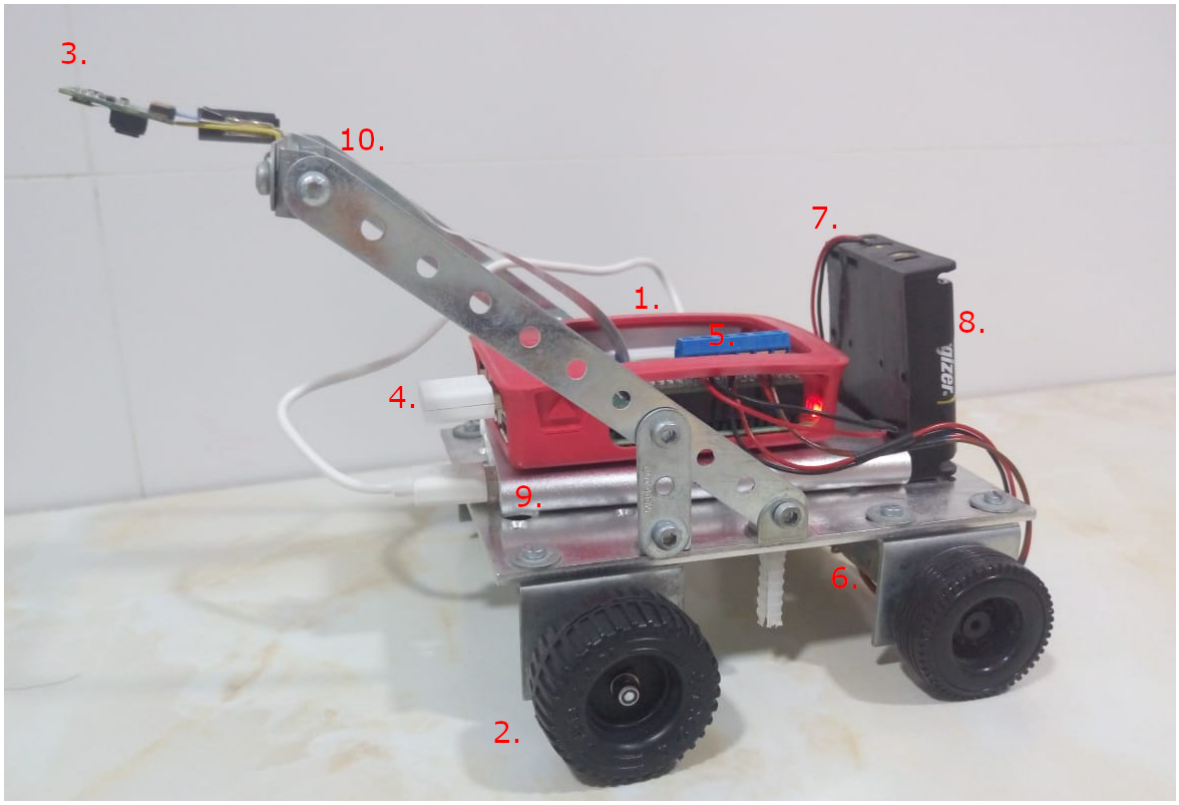


Figura 1: Prototipo construido

Componentes:

1.- Raspberry Pi 2 Modelo B (2015) con cubierta de plástico.

2.- 4 Ruedas de plástico

3.- Raspberry Pi Camera Module (5Mpx)

4.- Raspberry Wifi Dongle

5.- Pololu DRV8835 Dual Motor Driver Kit (H bridge)

6.- Dos servomotores

7.- Portapilas

8.- 4 pilas AA

9.- Power Bank usb (5000mAh)

10.- Soporte de Meccano

3.2.Software

En esta sección se detalla la arquitectura software del prototipo, de todos los sistemas que lo componen y las interacciones entre éstos.

Se pueden distinguir tres componentes:

- Navegador Web, ejecutado desde cualquier dispositivo.
- Raspberry
- Servidor, almacena y consulta los mapas.

El primer componente implementa un navegador web y es el encargado de la interacción con el usuario.

Desde cualquier dispositivo conectado a la misma red local que la Raspberry se puede acceder a la aplicación de control remoto del coche a través de un navegador Web. En la interfaz se determinan los comandos a ejecutar, ya sean de movimiento o búsqueda y se muestra el resultado obtenido de ésta última.

El componente central del sistema, la Raspberry, consta del módulo de "Raspberry Wifi Dongle" como adaptador Wi-Fi para permitir la conexión con el resto de subsistemas.

En primer lugar, aloja el servidor HTTP Apache, el cual tiene publicada la aplicación Web. Se decidió no realizar una apertura de puertos para que ésta solo sea accesible desde la red local y así evitar posibles conexiones no deseadas.

Además, se encarga de manejar todos los actuadores y sensores del coche, es decir, los motores y la cámara. Transforma los comandos de movimiento recibidos desde el cliente a instrucciones python para accionar los motores gracias al módulo "Motors.py" implementado.

El comando de búsqueda hace uso de la librería de python "Picamera" para realizar una captura al suelo.

Por último, este sistema envía las fotografías tomadas al servidor para después recibir el resultado en forma de coordenadas.

El último componente se trata un servidor externo. Se precisó de su uso debido a que la reducida capacidad de procesamiento de la Raspberry penalizaba mucho el procesamiento de imágenes y la búsqueda de las coordenadas en el mapa.

En esta máquina se ha establecido un socket escuchando, a la espera de recibir las imágenes con las que realizar la búsqueda en los mapas almacenados (data.npy, points.npy).

Una vez halladas las coordenadas correspondientes al frame recibido se envían como respuesta a la aplicación Web.

En la siguiente figura se muestra un esquema de los sistemas mencionados con el flujo de datos existente entre ellos.

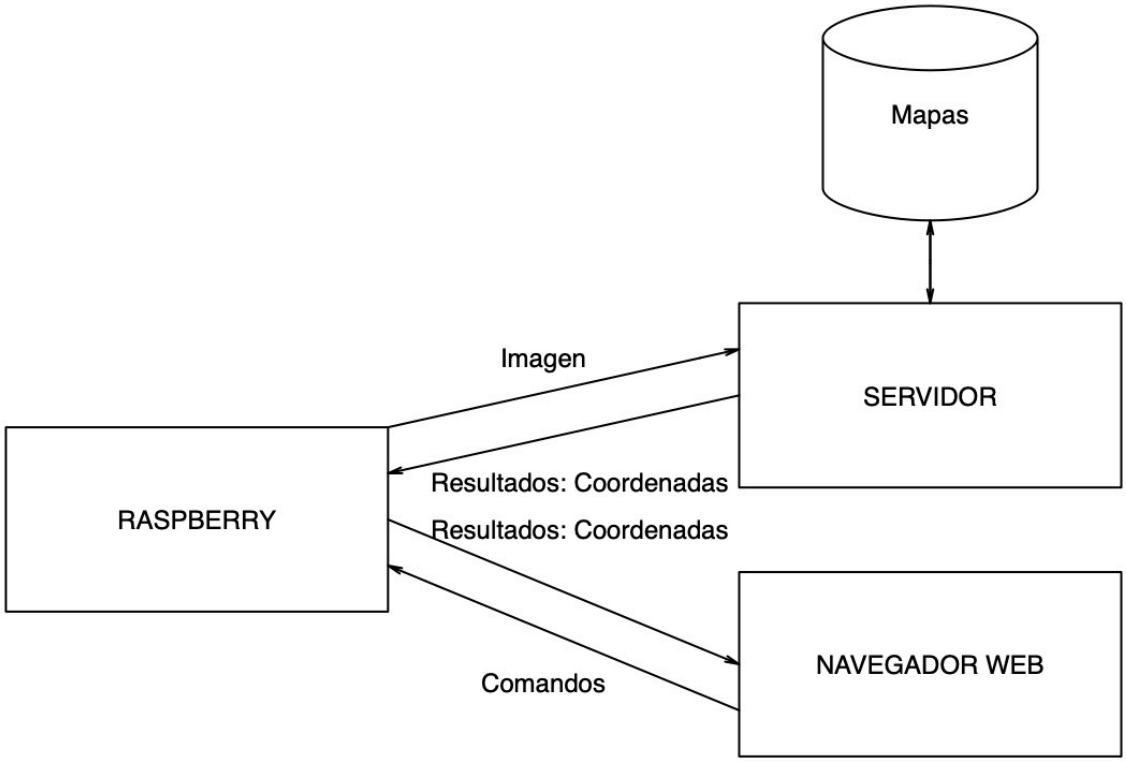


Figura 2: Diagrama Arquitectura Software

2.2.1 Secuencias

2.2.1.1 Secuencia de generación del mapa

En la siguiente figura se muestra el diagrama de secuencia que tiene lugar durante la generación del mapa.

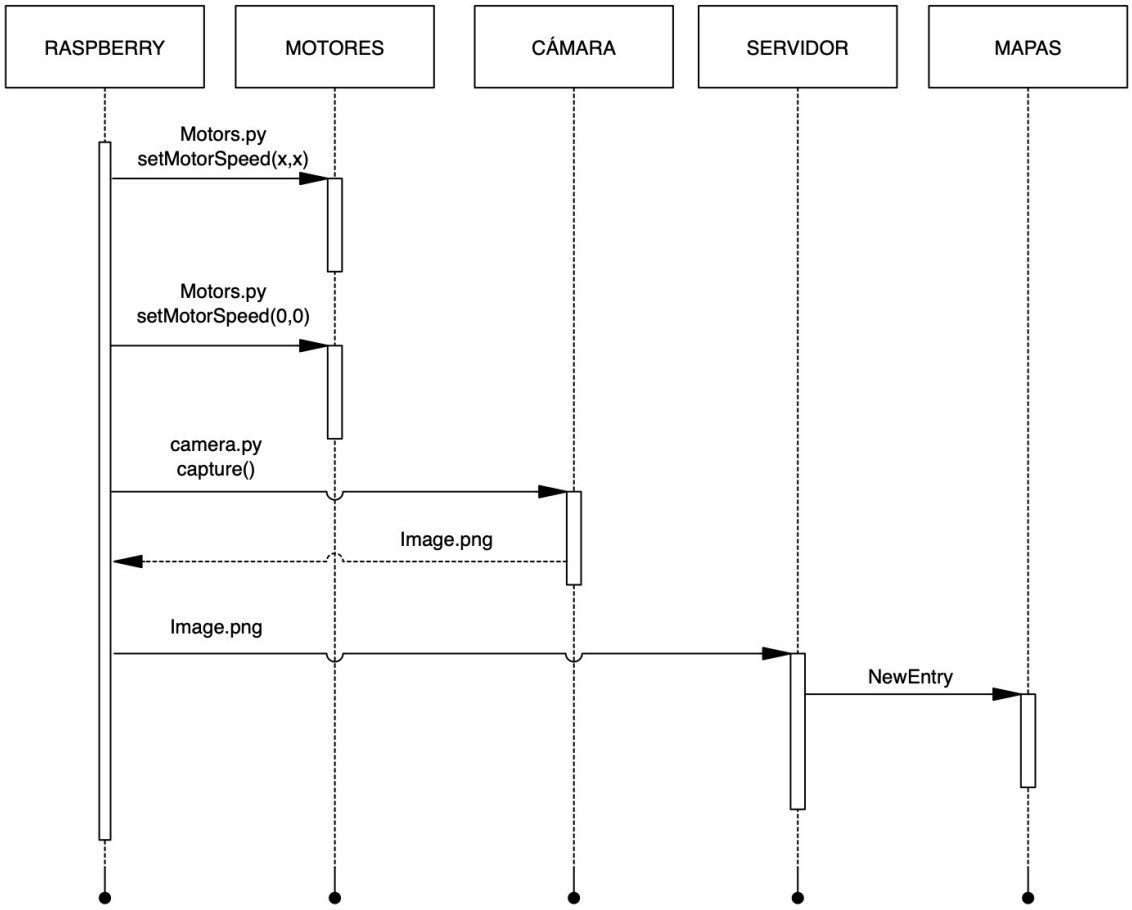


Figura 3: Diagrama Secuencia Generación de mapa

El robot se irá desplazando sobre el mapa, haciendo un barrido de este. Cada vez que se posicione sobre un área nueva deberá pausar los motores, con el objetivo de ganar una mayor estabilidad en al cámara. Realizará una captura y la enviará al servidor.

Una vez ahí, se lanzará el preprocesado de la imagen y la extracción de puntos, los cuales serán registrados en el mapa para sus posteriores consultas.

2.2.1.2 Secuencias de movimiento y búsqueda

A continuación se muestra un diagrama con las interacciones producidas entre los distintos sistemas y los componentes hardware a lo largo de los procesos de manejo del coche y localización en el mapa.

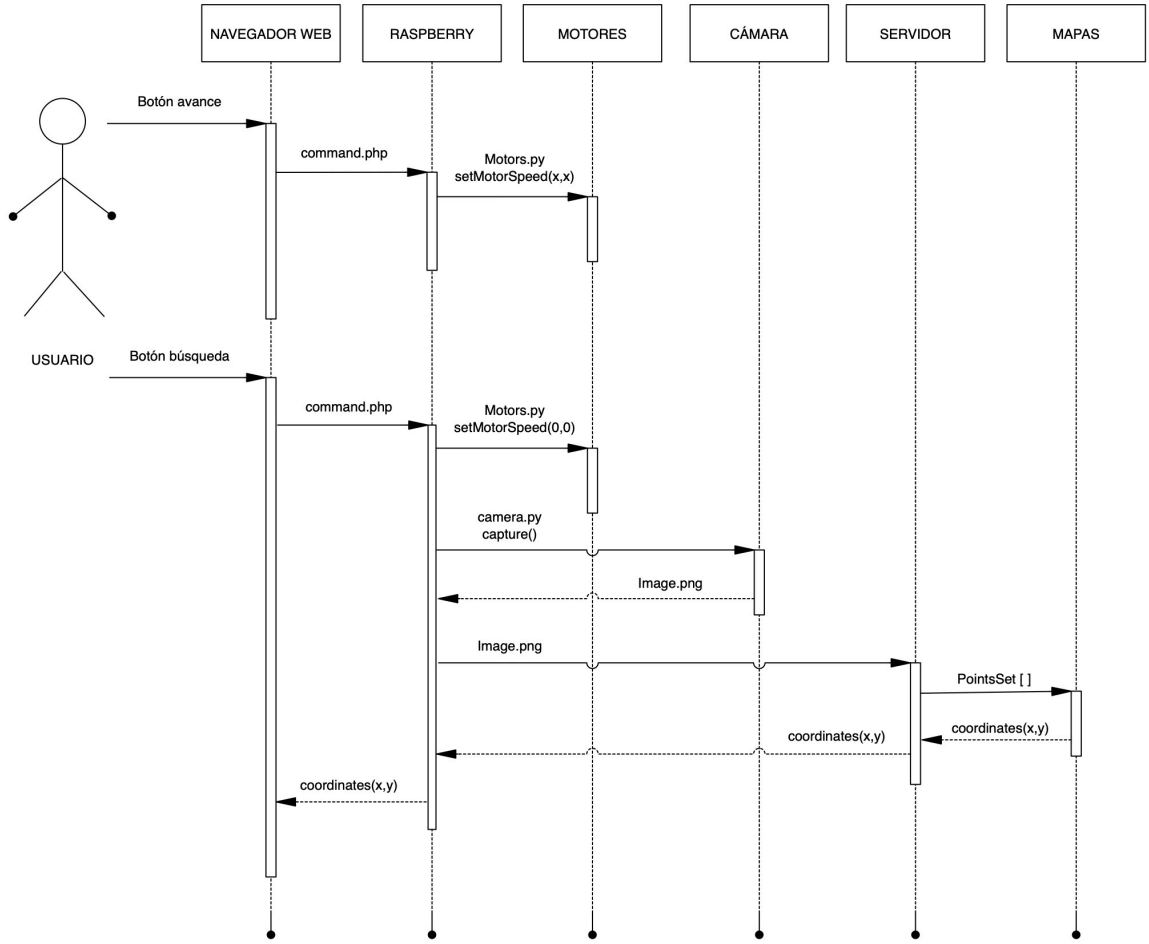


Figura 4: Diagrama Secuencia Manejo Robot

-Manejo del coche: el usuario acciona el botón de avance desde su navegador web y se manda el comando seleccionado a la Raspberry. Se ejecuta el módulo “motors.py” el cual accionará los motores con una misma velocidad y sentido.

-Localización: da comienzo cuando el usuario envía la orden por medio del botón de búsqueda de la aplicación. El comando es recibido por la raspberry, la cual se encarga de administrar los componentes hardware y de procesar la imagen. En primer lugar bloqueará los motores para lograr una mayor estabilización en la cámara y después ordenará la captura. Ésta se guardará en la memoria de la CPU y se enviará al servidor.

Es en este punto en el que tiene lugar el groso del algoritmo. Se realiza un preprocesado de la imagen con el módulo “preprocess.py", con el fin de aislar la información relevante y así poder extraer posteriormente los puntos característicos con mayor facilidad.

A continuación, con el módulo “areas.py" y la librería OpenCV se obtiene una matriz de puntos característicos de la imagen.

Con este set de puntos se realizará la consulta sobre los mapas y se obtendrán las coordenadas asociadas.

Finalmente, éstas son devueltas a la Raspberry y a la aplicación, que las mostrará por pantalla.

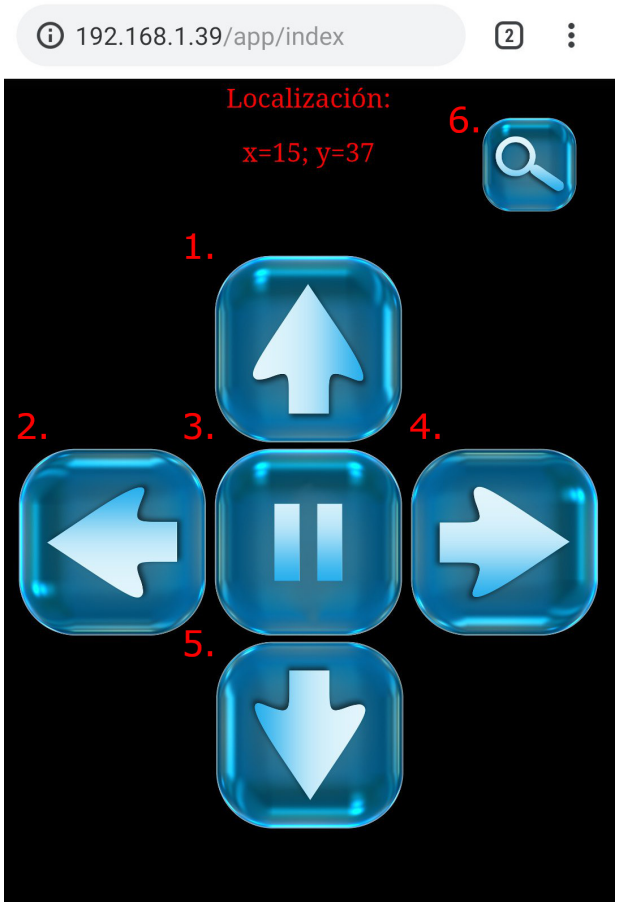
4.Documentación técnica del Software

En este apartado se detallan los distintos módulos desarrollados que componen el proyecto, incluyendo los procesos y algoritmos codificados en cada uno, así como las modificaciones que se tuvieron que llevar a cabo respecto al planteamiento inicial para solucionar las dificultades encontradas.

4.1. Aplicación Web

Ante la problemática surgida para controlar el aparato sin poder utilizar ningún dispositivo conectado por cable, dado que este disminuiría la movilidad y libre manejo, se decidió programar una simple aplicación web, accesible desde cualquier máquina con conexión a internet.

Se trata de una interfaz programada en html/php, con la que poder dirigir las operaciones necesarias para el correcto manejo del coche.



Los comandos configurados son:

1.- Avance

2.- Giro a la izquierda

3.- Pausa

4.- Giro a la derecha

5.- Retroceso

6.- Captura y localización en el mapa

Figura 5: Aplicación Web para el control del robot

Los botones de movimiento responden a una única pulsación, no es necesario mantenerlos presionados para continuar la acción. Cualquier interacción futura detendrá la acción anterior y comenzará con la nueva solicitada.

4.2. Control de Motores

Para el correcto manejo de los motores se ha desarrollado un módulo en python (motors.py).

Este programa se ejecuta en la Raspberry y se le ha dotado de los permisos necesarios para controlar los gpio.

Haciendo uso de la librería del puente H de Pololu y recibiendo por parámetro el comando seleccionado en la aplicación Web se puede fijar la velocidad y sentido de las dos ruedas motrices .

Los cinco casos contemplados permiten desplazar al robot en todas direcciones:

- **Avance:** se establece una velocidad de ambos motores de 300 rpm.
- **Giro a la derecha:** velocidad de 400 rpm en la rueda izquierda mientras que la derecha se fija a 400 rpm en sentido contrario.
- **Giro a la izquierda:** mismo caso anterior pero con inversión de los sentidos.
- **Marcha atrás:** -300 rpm en los dos motores.
- **Pausa:** paralización de los servos.

4.3. Cámara

Como sensor óptico se ha utilizado el módulo picamera en su versión 1.3 de 5 megapixeles. Este periférico se conecta a la Raspberry por medio del puerto CSI (Camera Serie Interface) y se sujeta con un soporte montado con piezas de meccano elevándolo a aproximadamente 14cm del suelo.

Para el correcto funcionamiento de este dispositivo se ha desarrollado un pequeño módulo en python (camera.py), el cual permite realizar capturas con una resolución de hasta 2592x1944 y guardarlas en la memoria de la CPU. Para este proyecto se ha decidido trabajar con imágenes de dimensiones 1256x800 puesto que facilitaban la ejecución de los algoritmos.

Estos frames, a la distancia a la que está situada la cámara del suelo, capturan un área real de aproximadamente 14cm de ancho por 7cm de alto.

Cabe destacar que como primera opción se pretendía elaborar el mapa a partir de una única captura mucho más alejada, tomada con otro dispositivo con mayor resolución. Pero después de numerosas pruebas y configuraciones fallidas no fue posible obtener puntos característicos similares entre el mapa y las capturas generadas por el robot, debido a las diferentes especificaciones de cada cámara. Es por esto por lo que se optó por cambiar el planteamiento inicial y elaborar el mapa mediante un barrido de la zona, capturando las imágenes con la misma cámara y a la misma distancia a la que luego se realizarán las búsquedas.

Tanto para la generación del mapa como para el proceso de búsqueda estas capturas se van sobrescribiendo con el fin de no saturar el disco duro del robot. Según se realiza la captura se envía al servidor y se extraen sus puntos, por lo que su almacenamiento no es necesario.

Uno de los mayores problemas encontrados fue calibrar el enfoque de la cámara. Ésta cuenta con un enfoque manual (se debe girar la lente un número de grados concreto según la distancia a la que se encuentre del objetivo), por lo que lograr el ajuste que mejor se adaptase al posterior algoritmo precisó de muchas pruebas e intentos.

4.4.Preproceso de Imágenes

Todas las imágenes capturadas, tanto las utilizadas para la elaboración del mapa como las de localización, son tratadas con el objetivo de aislar la información que nos resulte más relevante.

Con la ayuda de la librería de openCV se ha realizado un programa de procesamiento de imágenes, sin el cual los algoritmos de búsqueda y localización no funcionarían correctamente.

En primer lugar leemos la imagen y tratamos de reducir las sombras existentes. Para ello separamos los tres canales de color (RGB) y realizamos una diferencia entre cada uno de ellos y el resultado obtenido después de aplicar una dilatación y un filtro de mediana. Después volvemos a combinar los tres canales.

En el siguiente paso transformamos la imagen obtenida en el proceso anterior a escala de grises para limitar el rango de valores de luminosidad.

La variación de la luminosidad con la que se capturaba cada imagen fue un problema durante el desarrollo del algoritmo, es por ello, por lo que además del paso anterior, se realiza una corrección gamma, con el objetivo de reducir el impacto de este problema.

A continuación aplicamos un filtro gaussiano, para reducir el ruido existente y suavizar los contornos.

En este punto elaboramos el histograma de la imagen, el cual utilizaremos para acotar los valores hasta el umbral escogido. Esto nos permitirá aislar los puntos negros característicos de la imagen.

Para finalizar se realiza una umbralización sobre el histograma obtenido en el anterior paso, en nuestro caso aplicamos un umbral del 5%.

Por último se exporta la imagen resultante.

En las siguientes capturas se muestra el antes y el después del preproceso sobre una imagen tomada con el módulo de cámara de la Raspberry Pi. En ella se aprecia como se han podido aislar las áreas negras más características de la imagen

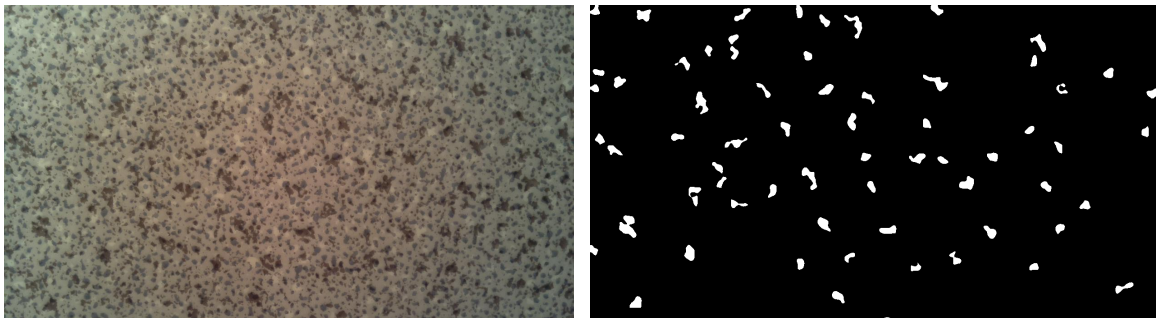


Figura 6: Transformación digital de una captura

Los parámetros utilizados en la dilatación, el filtro de mediana, la corrección gama y el filtro gaussiano dependen de la superficie que se esté capturando, por lo tanto el algoritmo deberá ser configurado en función del mapa a analizar.

4.5.Proceso y Algoritmo de Generación del Mapa

El primer paso a seguir para la elaboración de nuestro mapa se basa en un barrido del coche sobre la superficie que deseamos registrar. Esta no era la primera opción que se contempló, pues se intentó generar el mapa a partir de una única imagen que recogiera toda la superficie, pero por las diferentes características de la cámara y la lejanía al objetivo no se obtenían las mismas áreas.

Con el planteamiento escogido, el dispositivo se desplazará tomando capturas, asignándoles un identificador único y almacenándolas en su memoria interna. Una vez finalizado este barrido se dará paso al preproceso, citado en el apartado anterior, y al algoritmo de obtención de puntos para cada imagen.

El objetivo del algoritmo consiste en mapear todos los puntos característicos de las imágenes resultantes en dos estructuras diccionario de python, las cuales serán guardadas en el disco duro del servidor.

El programa comienza con una selección de puntos o áreas que cumplan con un tamaño / masa mínima. Para lograr esto se emplean las funciones ‘findContours’ y ‘Moments’ que nos proporciona la librería de OpenCV. Cabe destacar que el límite de tamaño de área deberá ser parametrizado atendiendo a las características de la superficie con la que se está trabajando, esto permitirá cribar la información no deseada.

En el caso de que el número de puntos encontrados no llegara a 12, esa imagen no se tendría en cuenta en la elaboración del mapa ya que no contendría suficiente información característica.

Después de realizar esta selección se buscarán para cada una de las áreas encontradas los 12 puntos más cercanos pertenecientes a los escogidos anteriormente, a los que llamaremos puntos vecinos, conformando así una serie de, como denominaremos a partir de ahora, constelaciones.

Además, estos vecinos deben encontrarse entre dos distancias parametrizadas, una mínima y una máxima, desde el centro de la constelación. Este filtro es necesario para no tener en cuenta los vecinos demasiado próximos, cuyo ángulo formado con el centro podría sufrir alteraciones. Tampoco es conveniente utilizar vecinos demasiado lejanos, pues la probabilidad de que se salgan de la imagen capturada por el robot aumentaría.

En la siguiente imagen se muestra un ejemplo de constelación, con su área centro y sus 12 vecinos.

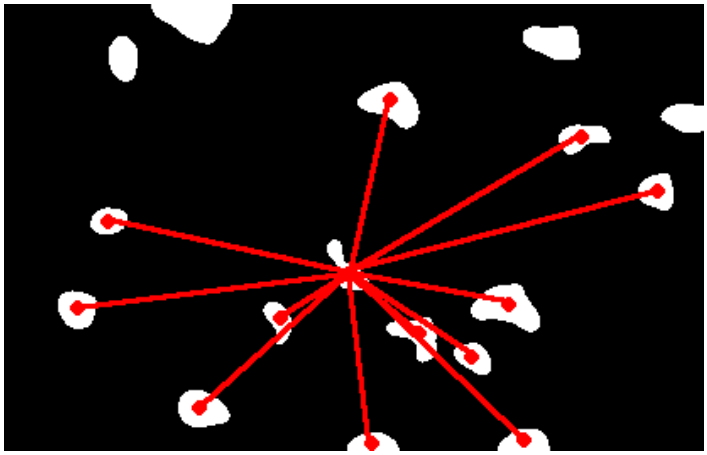


Figura 7: Constelación localizada en imagen procesada

Como último paso faltaría salvar toda esta información en un diccionario de Python. Para ello, por set de tres puntos (centro de la constelación y 2 vecinos), se crea una entrada en el diccionario compuesta de:

- Las coordenadas del centro de la constelación
- Las coordenadas de los dos vecinos
- Los tres ángulos formados por estos 3 puntos
- El identificador de la imagen a la que pertenecen
- Distancia al centro de la constelación

Obteniendo de esta forma 12 entradas de 11 posiciones por cada constelación.

Para establecer el índice de elemento del diccionario (el cuál funciona como una tabla hash), se aplica la siguiente fórmula:

$$índice = \alpha + \beta * 360$$

Siendo α el ángulo formado por el centro y sus dos vecinos tomando el centro como vértice y β considerando el vecino 2 como vértice.

Además, se genera otro diccionario a modo de apoyo, el cual almacenará información relativa a cada punto o área.

La clave de éste estará compuesta por el identificador de la imagen a la que pertenece y las coordenadas del centro.

Los valores guardados serán:

- Las coordenadas del centro del punto
- El ángulo que forma éste con el centro y el eje X
- La masa del centro

Finalmente, ambos diccionarios son guardados como ficheros binarios, en los que se irá concatenando la información correspondiente a cada imagen procedente del barrido.

4.6. Algoritmo de Búsqueda

En un principio se pretendía que el trabajo del algoritmo de búsqueda se repartiese entre dos dispositivos, la Raspberry y el servidor. Pero debido a los insuficientes recursos (memoria y procesador) de los que dispone la Raspberry se tuvo que traspasar enteramente al servidor, y de esta forma minimizar los tiempos de respuesta.

En primer lugar, se busca el punto más céntrico de la imagen, el cual debe cumplir el mismo tamaño mínimo fijado en el algoritmo de elaboración del mapa.

A partir de este centro se buscan los 12 vecinos más cercanos, cumpliendo las distancias mínima y máxima configuradas, ordenados por el ángulo que forman con el centro y la horizontal, repitiendo así el proceso llevado a cabo en el anterior algoritmo.

Con la constelación resultante se dará lugar a la búsqueda en el primer diccionario, obteniendo previamente el índice, con la fórmula ya descrita, para cada punto de la constelación.

Dado que en el diccionario existirán *colisiones*, se creará una lista con todos los posibles identificadores de puntos.

Se consultará el segundo diccionario y, por cada entrada encontrada, se procederá a comparar las constelaciones de los centros posibles con la analizada por medio de sus ángulos y la masa del centro.

Se crearán dos arrays de 360 posiciones, marcando como bit activo las posiciones/ángulos en los que exista algún punto. Después se hará rotar uno de estos arrays buscando el menor número de diferencias. Si se superan 6 coincidencias de bits activos, de los 12 posibles, se marcará ese centro como válido.

El motivo de esta rotación se debe a que no todas las capturas del robot tendrán la misma orientación que las imágenes que componen el mapa, por eso el trabajo se realiza con los ángulos, elemento que se mantiene invariable.

En la siguiente figura se muestra un ejemplo de la rotación de dos Arrays acortados pertenecientes a constelaciones semejantes pero entre los que existe una diferencia de 2 grados en la orientación.

Figura 8: Ejemplo de rotación de Arrays de ángulos

Finalmente se escogerá el ID más repetido de entre las comparaciones correctas y se devolverán las coordenadas absolutas correspondientes al centro de la imagen localizada a la aplicación Web, para que éstas sean mostradas por interfaz al usuario.

5.Integración, pruebas y resultados

A continuación se detalla la batería de pruebas, tanto unitarias como en conjunto, que se han llevado a cabo para comprobar el correcto funcionamiento de todos los módulos del robot.

Se han realizado pruebas del control remoto de los motores. El tiempo de respuesta obtenido ha sido óptimo y el manejo fluido, tanto con conexión wifi como con conexión 3G. Cualquier parte del mapa es accesible con un número reducido de movimientos.

Existe un pequeño fallo hardware que provoca que en ciertas ocasiones la rueda motriz izquierda experimente un menor rozamiento con el suelo, lo que da lugar a dificultades principalmente en el giro hacia la derecha. Pero teniendo en cuenta que este era un error mínimo y no fundamental para el objetivo del proyecto se ha preferido destinar más tiempo en subsanar otro tipo de fallos.

La captura de imágenes y el envío de éstas al servidor son el subproceso más lento de toda la cadena. Toma unos tiempos de ejecución de aproximadamente 3 segundos, a pesar de que se redujo la calidad de las fotos hasta lo estrictamente necesario.

La mayor parte de las pruebas se han centralizado sobre el proceso de localización, dado que es la base de este trabajo.

En primer lugar se obtuvo una colección de 1600 frames distintos de la superficie escogida y se generó el mapa.

Utilizando cada una de éstas imágenes se lanzó el proceso de localización, obteniendo identificaciones satisfactorias con un 100% de acierto.

En la siguiente prueba se realizó una rotación aleatoria de estas imágenes y se volvió a lanzar el proceso de localización para cada una de ellas. Los resultados también fueron exitosos en todos los casos.

Se realizaron combinaciones de imágenes, es decir, frames formados por distintas partes de las anteriores 1600 imágenes capturadas. El desenlace cumplió con lo esperado, el algoritmo localizaba el frame en el que se encontraba el centro de la imagen analizada.

Finalmente se llevaron a cabo pruebas en conjunto de todos los módulos.

Una vez generado un mapa delimitado, se lanzaron varias pruebas de localización. Hasta un 80% resultaron exitosas. La sombra proyectada por el propio robot, que altera los puntos característicos de las imágenes y las diferencias en la incidencia e intensidad de la luz explican las localizaciones no encontradas en algunos casos, pero estas suelen corregirse con un segundo intento de búsqueda.

A pesar del tratamiento previo de los frames, en los que se minimizan las sombras y se trabaja la iluminación, no se ha conseguido llegar a la configuración exacta con la que el algoritmo deje de ser sensible a estos factores.

En el siguiente link se adjuntan pruebas gráficas del funcionamiento del robot, en las que se demuestra que es capaz de localizar en entornos ideales, con focos que minimicen todo lo posible los agentes externos que pudieran interferir y elaborando el mapa instantes antes de la búsqueda.

Vídeo demostrativo: https://www.youtube.com/watch?v=MPzwdbrkOWk

6.Conclusiones y trabajo futuro

6.1.Conclusiones

Este proyecto ha supuesto un gran reto debido a su dificultad, pero me ha hecho aprender y crecer como ingeniero.

He logrado manejar un nuevo lenguaje para mi, como es Python y he descubierto el abanico de posibilidades que te ofrece una Raspberry Pi y la librería OpenCV.

A pesar de no haber llegado al 100% del objetivo buscado, debido a la sensibilidad del robot, creo que he alcanzado una aproximación bastante meritoria. El robot es capaz de identificar su localización en un mapa pequeño y bajo condiciones ideales de luces y sombras.

Soy consciente de que este trabajo podría tener una gran evolución en el futuro, una versión 2 del robot podría corregir todos los fallos y carencias de este prototipo, por ello animo a que otro alumno continue con mi labor perfeccionándola.

6.2.Trabajo futuro

Tomando como base el actual prototipo construido podrían llevarse a cabo una serie de mejoras y optimizaciones que dotarían al robot de una mayor eficiencia en su labor.

Hardware

Una renovación de la picamera por su último modelo (V2.1), mejoraría la resolución de las fotografías tomadas. Bien es cierto que el nivel de detalle en las capturas no es una de las necesidades de este proyecto, pues tan solo son esenciales los puntos característicos, pero con otro diseño hardware y un mejor estabilizador de cámara podrían lograrse capturas más nítidas en movimiento, lo que no obligaría a paralizar el robot en el proceso de localización.

La robustez y estética del coche cuentan con un gran margen de mejora. Una carcasa que protegiese la CPU, las baterías y el cableado serían indispensables en una versión final del coche.

Software

A pesar del tratamiento a que se someten las imágenes no se han logrado eliminar todas las sombras y ruido ni se ha corregido en su totalidad la iluminación. Una revisión y recodificación del proceso podría minimizar estos fallos y con ello mejorar la capacidad de localización del robot.

También se plantea la opción de instalar unos leds a modo de faros que iluminen la zona a capturar. Esto dotaría al coche de una mejor adaptabilidad a los distintos entornos en los que puede trabajar.

Referencias

1. www.raspberrypi.org [Último Acceso 30/10/2018]
2. www.python.org [Último Acceso 15/02/2019]
3. www.pololu.com [Último Acceso 23/09/2018]
4. https://docs.opencv.org [Último Acceso 10/12/2018]
5. https://docs.scipy.org/doc [Último Acceso 25/02/2019]
6. https://stackoverflow.com/questions/19909167/how-to-find-most-frequent-string-element-in-numpy-ndarray [Último Acceso 10/12/2018]
7. https://www.jeffgeerling.com/blog/2017/fixing-blurry-focus-on-some-raspberry-pi-camera-v2-models [Último Acceso 20/09/2018]
8. https://geekytheory.com/tutorial-raspberry-pi-uso-de-picamera-con-python [Último Acceso 24/09/2018]
9. www.agps.es [Último Acceso 10/03/2019]
10. http://www.car.upm-csic.es/lopsi/static/publicaciones/docencia/Apuntes%20RF-LPS.pdf [Último Acceso 17/03/2019]
11. http://oa.upm.es/947/1/PFC_LUIS_DIAZ_AMBRONA.pdf [Último Acceso 17/03/2019]
12. Universidad Autónoma de Madrid. Method for positioning devices relative to a surface, ES2566427A1. [Publicación 12-04-2016]

Glosario

Raspberry Pi Ordenador de placa reducida con fines principalmente educativos.

Diccionario Estructura de datos que asocia llaves o claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos almacenados a partir de una clave generada.

Colisión Situación que se produce cuando dos entradas distintas a una función de hash producen la misma salida

Librería Conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

Socket Mecanismo de intercambio de datos entre dos programas.

Puente H Circuito electrónico que permite girar en ambos sentidos a un motor eléctrico.

Prototipo Primer ejemplar que se fabrica de una figura o invento.

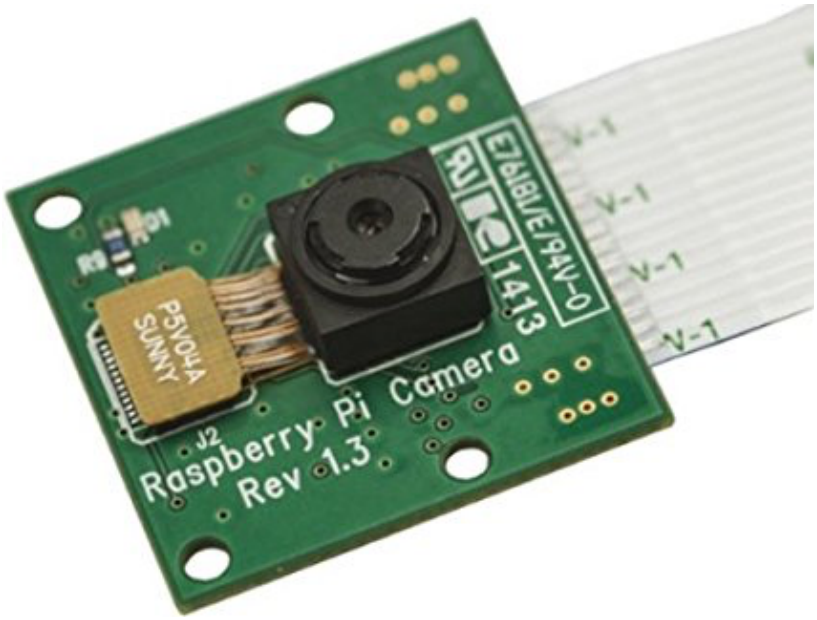
Frame Fotograma, instantánea.

GPIO General Purpose Input/Output

Anexos

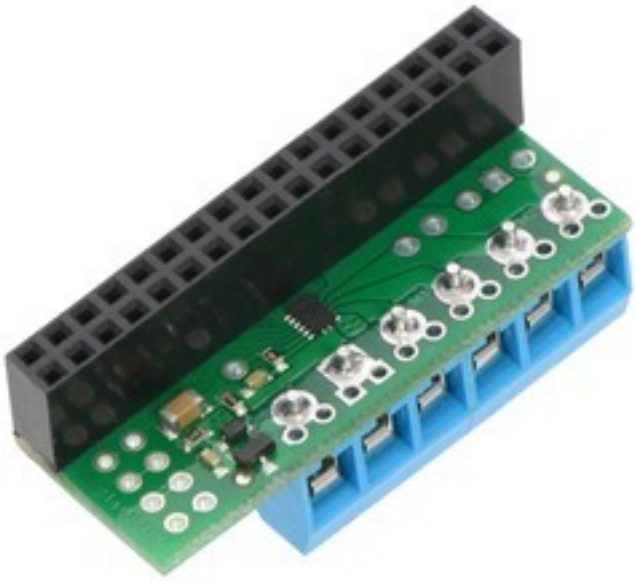
A. Hardware

Raspberry Pi 2 Modelo B



Pi Camera V1.3 5Mpx

Raspberry Wifi Dongle



Pololu DRV8835 Dual Motor

B. Software

Index.html

```
<html>
     <head>
          <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/
jquery.min.js"></script>
          <link rel="stylesheet" type="text/css" href="style.css">
     </head>
     <body>
          <div class="wrapper" style="text-align:center;">
               <button id="buscar"><img src="/images/search.png"></button>
               <label id="localizacion">Localización: </label>
               <label id="resultado">x=0, y=0 </label>
               <br>
               <button id="avanzar"><img src="/images/up.png"></button>
               <br>
               <button id="derecha"><img src="/images/left.png"></button>
               <button id="pausar"><img src="/images/pause.png" ></button>
               <button id="izquierda"><img src="/images/right.png"></button>
               <br>
               <button id="retroceder"><img src="/images/down.png"></button>
          </div>
     </body>
     <script>
          $(document).ready(function(){
              $("#avanzar").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:{"action":"w"}
               });
              });
              $("#retroceder").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:{"action":"s"}
               });
              });
              $("#derecha").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:{"action":"d"}
               });
              });
              $("#izquierda").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:{"action":"a"}
               });
              });
              $("#pausar").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:{"action":"p"}
               });
              });
              $("#buscar").click(function(){
               $.ajax({url: "executeCommand.php",type:'post',data:
{"action":"search"},
               success: function(data){
                    document.getElementById("resultado").innerHTML = data;
               }
               });
              });
          });
     </script>
</html>
```

Motors.py

```
from pololu_drv8835_rpi import motors, MAX_SPEED
import sys

if sys.argv[1]=="w":
    motors.setSpeeds(-300, -300)

elif sys.argv[1]=="s":
    motors.setSpeeds(300, 300)

elif sys.argv[1]=="a":
    motors.setSpeeds(400,-400)

elif sys.argv[1]=="d":
    motors.setSpeeds(-400,400)

elif sys.argv[1]=="p":
    motors.setSpeeds(0,0)
```

Preprocess.py

```
import io
import time
import cv2
import numpy as np
import sys

def adjust_gamma(image, gamma=1.0):

   invGamma = 1.0 / gamma
   table = np.array([((i / 255.0) ** invGamma) * 255
      for i in np.arange(0, 256)]).astype("uint8")

   return cv2.LUT(image, table)

def processImage():

    #Remove shadows
    img = cv2.imread('capture.png', -1)

    rgb_planes = cv2.split(img)

    result_planes = []
    result_norm_planes = []
    for plane in rgb_planes:
        dilated_img = cv2.dilate(plane, np.ones((21,21), np.uint8))
        bg_img = cv2.medianBlur(dilated_img, 21)
        diff_img = 255 - cv2.absdiff(plane, bg_img)
        result_planes.append(diff_img)

    result = cv2.merge(result_planes)
    cv2.imwrite('capture1.png', result)

    image= cv2.imread('capture1.png',0)

    #Gamma Adjust
    gamma = 0.5
    adjusted = adjust_gamma(image, gamma=gamma)

    #Gaussian Filter
    im_gaussian = cv2.GaussianBlur(adjusted, (50,50),0)
```

```
    #Generate histogram
    height, width = im_gaussian.shape;
    h= [0] * 256;
    ha = h
    for i in range(0,height):
        for j in range(0,width):
            h[im_gaussian[i][j]] += 1;

    for z in range(1,256):
        ha[z]=ha[z-1]+h[z];

    umbral=ha[255] * 0.05;
    for ui in range(0,255):
        if ha[ui] >= umbral:
            break;

    #Thresholding
    ret,thresh  = cv2.threshold(im_gaussian, ui, 255, cv2.THRESH_BINARY_INV)

    nb_components, output, stats, centroids =
cv2.connectedComponentsWithStats(thresh , connectivity=8)
    sizes = stats[1:, -1]; nb_components = nb_components - 1
    min_size = 250
    contador = 0
    img2 = np.zeros((output.shape))
    for i in range(0, nb_components):
        if sizes[i] >= min_size:
            img2[output == i + 1] = 255
            contador += 1

    #Result
    cv2.imwrite('processed.png', img2)
```

Funciones Areas.py

```
#Carga de Diccionarios
def loadArray(path):
    return np.load(path).item()



#Búsqueda de áreas con un tamaño mínimo S
def find_connected_components(img,S):
    _, contour, _= cv2.findContours(img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    centers = np.zeros(shape=(len(contour),3))
    n=0
    for i in range(len(contour)):
      M = cv2.moments(contour[i])
      if M['m00']> S :
            centers[n]=[int (M['m10']/M['m00']),int (M['m01']/
M['m00']),int(M['m00'])]
            n = n+1
    return centers[:n]


#Búsqueda del punto más cercano al centro
def find_center_point(centers,dimensionX,dimensionY):
    distance=dimensionX;
    for i in range(len(centers)):
        dis= calculate_distance(dimensionX/2,dimensionY/2,centers[i,0],centers[i,
1])
        if  dis < distance :
                distance=dis
                center=centers[i]
    return center
```

```
#Cálculo de distancia entre dos puntos
def calculate_distance(x1,y1,x2,y2):
    dist = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return dist


#Búsqueda de los 12 vecinos más cercanos al centro
#Definiendo una distancia minima y máxima
def find_neighbours(centers,center_point,r1,r2):
    aux = []
    for i in range (len(centers)):
        dist=calculate_distance((center_point[0]),(center_point[1]),(centers[i,
0]), (centers[i,1]))
        if dist > r1 and dist < r2:
            aux.append((centers[i,0],centers[i,1],dist))
    neighbours = np.array(aux)
    neighbours = neighbours[neighbours[:,2].argsort()]
    return neighbours[0:12, :]


#Calcula el ángulo entre dos puntos y el eje X (radianes)
def get_angle_between_points(x_orig, y_orig,x_landmark, y_landmark):
    deltaY = y_landmark - y_orig
    deltaX = x_landmark - x_orig
    rads = math.atan2(deltaY,deltaX)
    return rads


#Transforma un ángulo de radianes a grados
def angle_trunc(a):
    while a < 0.0:
       a += 2*math.pi
    a=math.degrees(a)
    while a > 360:
       a-=360
    return a


#Búsqueda del ángulo que forman los vecinos con el eje X y ordenación por este
def find_neighbours_angles(center_point,neighbours):
    angles = np.zeros(shape=(len(neighbours),4))
    for i in range(len(neighbours)):
         angles[i]=[neighbours[i,0],neighbours[i,1],('%.2f' % (360-
angle_trunc(get_angle_between_points(float (center_point[0]),float
(center_point[1]),float (neighbours[i][0]),float (neighbours[i]
[1])))))),neighbours[i,2]]

    angles = angles[angles[:,2].argsort()]
    return angles


#Búsqueda en diccionario de constelaciones
def get_from_direct_array(alfa,beta,direct_array):
        return direct_array[alfa+360*beta]
```

```
#Obtiene los ángulos formados por dos vecinos y el centro
def get_triangles(center_point,angles):
      triangle_angles = np.zeros(shape=(len(angles),9))
      for i in range(len(angles)):
                if (i != (len(angles)-1)):
                        triangle_angles[i,0] = angles[i,0]
                        triangle_angles[i,1] = angles[i,1]
                        triangle_angles[i,2] = angles[i+1,0]
                        triangle_angles[i,3] = angles[i+1,1]
                        triangle_angles[i,4] =
round(angle_trunc((math.atan2(angles[i,1]-center_point[1],angles[i,0]-
center_point[0])-(math.atan2(angles[i+1,1]-center_point[1],angles[i+1,0]-
center_point[0])))))
                        triangle_angles[i,5] =
round(angle_trunc((math.atan2(center_point[1]-angles[i+1,1],center_point[0]-
angles[i+1,0])-(math.atan2(angles[i,1]-angles[i+1,1],angles[i,0]-
angles[i+1,0])))))
                        triangle_angles[i,6] =
round(angle_trunc((math.atan2(angles[i+1,1]-angles[i,1],angles[i+1,0]-angles[i,
0])-(math.atan2(center_point[1]-angles[i,1],center_point[0]-angles[i,0])))))

                else :
                        triangle_angles[i,0] = angles[i,0]
                        triangle_angles[i,1] = angles[i,1]
                        triangle_angles[i,2] = angles[0,0]
                        triangle_angles[i,3] = angles[0,1]
                        triangle_angles[i,4] =
round(angle_trunc((math.atan2(angles[i,1]-center_point[1],angles[i,0]-
center_point[0])-(math.atan2(angles[0,1]-center_point[1],angles[0,0]-
center_point[0])))))
                        triangle_angles[i,5] =
round(angle_trunc((math.atan2(center_point[1]-angles[0,1],center_point[0]-
angles[0,0])-(math.atan2(angles[i,1]-angles[0,1],angles[i,0]-angles[0,0])))))

                if triangle_angles[i,4] > 180 or triangle_angles[i,5] > 180:
                        triangle_angles[i,4]=360-triangle_angles[i,4]
                        triangle_angles[i,5]=360-triangle_angles[i,5]
                        triangle_angles[i,6]=360-triangle_angles[i,6]
                triangle_angles[i,7] = int(angles[i,2])
                triangle_angles[i,8] = int(angles[i,3])
      return triangle_angles


#Compara los ángulos de dos constelaciones
def compareConstels(actualConstel, mapConstel):
    coincidencias = 0
    array1 = np.zeros(shape=(360))
    array2 = np.zeros(shape=(360))
    #Activa los bits del array con ese ángulo en la constelación
    for i in range(len(mapConstel)):
        array1[(int)(mapConstel[i][2])-1] = 1
    for i in range(len(actualConstel)):
        array2[(int)(actualConstel[i][2])-1]=1
    for i in range(len(array2)):
        if (((array1 != array2).sum()) <= 12):
            return 1
        array2 = np.roll(array2,1)
    return 0
```

```
#Main generación del mapa
def main(vdirect_array,vpoint_array,ID):

    image= cv2.imread('Images/processed'+ID+'.png',0)

    centers=find_connected_components(image,300)
    if len(centers) < 12 :
        print('No se han encontrado suficientes areas')
        return

    for i in range(len(centers)):
        #Puntos vecinos ordenados por distancia al centro
        neighbours=find_neighbours(centers,centers[i],50,300)

        #Puntos vecinos ordenados por angulos
        angles=find_neighbours_angles(centers[i],neighbours)

        #Vecinos con alfa y beta
        triangle_angles=get_triangles(centers[i],angles)

        #Insercion en Diccionarios
        insert_all_points_direct_array(centers[i],triangle_angles, vdirect_array,
vpoint_array,ID)
```